HAAGA-HELIA
ammattikorkeakoulu

Modernin web-pohjaisen sovellusalustan jatkokehitys

Veli-Matti Luoto

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2014

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

29.04.2014

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Veli-Matti Luoto	**Ryhmätunnus tai aloitusvuosi** 2010
Raportin nimi Modernin web-pohjaisen sovellusalustan jatkokehitys	**Sivu- ja liitesivumäärä** 31 + 5
Opettajat tai ohjaajat Sirpa Marttila	

Opinnäytetyö käsittelee verkkopalveluna toimivan Deveo-sovellusalustan sovellustuen ja sovellusarkkitehtuurin jatkokehittämistä. Projektin asiakkaana ja tilaajana toimi Deveo Oy.

Projektin tehtävänä oli toteuttaa alustariippumaton komentorivityökalu Deveo-sovellusalustan sovellusten luomiselle ja paketoimiselle, tuki Deveo-liitännäisten kokonaisvaltaiselle käsittelylle ja tuki sovellusten käyttöönottoon Deveo-asennusinstansseissa.

Toteutetun produktin kehitysprosessia ja teknisiä ratkaisuja esitellään opinnäytetyössä yksityiskohtaisemmin. Produktin eri komponentit rakennettiin pienimmällä mahdollisella vaivalla siten, että toteutus silti vastasi toimeksiantajan asettamia laatuvaatimuksia. Loppukäyttäjän näkökulmasta toteutettu produkti ilmenee yksinkertaisena tapana luoda ja ottaa käyttöön Deveo-sovellusalustan sovelluksia Deveo-asennusinstansseissa.

Kokonaisuutena opinnäytetyöprojekti oli onnistunut ja sen tuloksena syntyi vaatimusmäärittelyn täyttävä produkti.

Asiasanat
Deveo, ohjelmistokehitys, REST, sovellusalusta

HAAGA-HELIA
University of Applied Sciences

Abstract

29.04.2014

Degree programme in Business Information Technology

Authors Veli-Matti Luoto	**Group or year of entry** 2010
The title of thesis The further development of a modern web-based application platform	**Number of report pages and attachment pages** 31 + 5
Advisor(s) Sirpa Marttila	

This thesis covers the further development of the web-based application platform Deveo and its application architecture. The thesis was commissioned by the project's client, Deveo Ltd.

The purpose of this thesis was to implement a platform-independent command line interface for creating and packaging Deveo application platform applications, support for all-encompassing management of Deveo plugins, and support for deployment of Deveo application platform applications in Deveo instances.

The development process of the product and technical solutions are presented in the thesis in more detail. The different components of the product were built by investing the minimum viable effort for developing an implementation where the client's qualitative requirements were met. From the end user perspective, the implemented product appears as a convenient way of creating and deploying Deveo application platform applications in Deveo instances.

Overall, the thesis project was a success and resulted in a product that fulfilled the specified requirements.

Key words
Application platform, Deveo, REST, software development

Sisällys

1 Johdanto

Deveo on suuryrityksille suunnattu koodinhallinta-alusta, jonka keskeisin toimintaperiaate on sen laajennettavuus. Deveon avulla kyetään pystyttämään ja hallinnoimaan ohjelmistoprojektin ekosysteemiä vaivattomasti. Deveo voidaan asentaa yrityksen omiin tiloihin tai sitä voidaan käyttää suoraan pilvestä. (Deveo 2014d.)

Deveo on Eficode Oy:stä lähtöisin oleva Spin-off -yritys, jolla oli opinnäytetyön tekohetkellä neljä työntekijää opinnäytetyön tekijä mukaan lukien. Vaikka yritys on pieni, on se silti monikansallinen. Työtä tuotteen eteen tehdään niin Suomesta kuin Espanjastakin käsin. Työkielenä yrityksessä on englanti, minkä vuoksi kaikki opinnäytetyössä esitellyt työtulokset ovat englanniksi.

Projektin aikana tuotettava produkti tullaan tekemään osaksi nykyistä verkkopalveluna toimivaa Deveo-sovellusalustaa. Deveo on alun perin suunniteltu laajennettavaksi sovellusalustaksi, mutta sen laajennettavuuden kannalta välttämättömien komponenttien ja työkalujen toteutus on jäänyt puolitiehen. Pääsyynä projektin käynnistämiselle on alkuperäisen vision, aidosti laajennettavan sovellusalustan, eteenpäin vieminen. Jonkinasteista pohjatyötä projektiin liittyen on jo etukäteen tehty. Projektin tavoiteltu hyöty on mahdollisimman vankka pohja Deveon sovellusarkkitehtuurille, jonka tarkoituksena on lopulta mahdollistaa yhteistyö toimeksiantajan ja sen ulkopuolisten toimijoiden kanssa.

Tämä opinnäytetyö käsittelee Deveo-sovellusalustan sovellustuen jatkokehittämistä. Opinnäytetyö toteutetaan produktityyppisenä työnä, jonka lopputuloksena syntyvät alustavat komponentit ja työkalut sovellusalustan sovellusten luomiselle, paketoimiselle ja käyttöönotolle Deveo-asennusinstansseissa. Toteutettaviin komponentteihin ja työkaluihin kuuluu komentorivityökalu sovellusten luomista ja paketointia varten, tuki sovellusalustan sovellusten käyttöönotolle ja tuki sovelluksia vastaavien Deveo-liitännäisten käsittelemiselle. Projektissa syntyvät komponentit ja työkalut tulevat olemaan osa nykyistä sovellusalustatoteutusta.

Aihe on alalla hyvin ajankohtainen, sillä jo nyt moni yritys hyödyntää sovellus- ja liitännäisarkkitehtuuria verkkopalveluissaan tarjoten palveluilleen lisäominaisuuksia sovellusten ja liitännäisten muodossa. Aihe valittiin opinnäytetyön toimeksiantajan johdosta sillä perustein, että sovellusalustaa saataisiin ideana vietyä pidemmälle. Opinnäytetyö on suunnattu web-sovelluskehityksestä kiinnostuneille ohjelmistoalalla opiskeleville ja työskenteleville henkilöille.

1.1 Aiheen rajaus ja haasteet

Projekti käsittelee sovellusalustan sovellusten käyttöönottoa Deveo-asennusinstanssissa ja sovellusalustaan tehtyjä muutoksia sen mahdollistamiseksi. Produktin osalta pois rajattuja osa-alueita ovat sovellusalustan sovellusten sovelluskehitysympäristön työkalujen ja sovellusten jakeluportaalin toteutus. Produktin komponentteja testataan projektin aikana toimeksiantajan määrittelemillä tavoilla, mutta testaus jätetään käsittelemättä tässä opinnäytetyössä.

Merkittävimmät haasteet liittyvät projektin ajanhallintaan, sillä opinnäytetyön tekijä teki muita toimeksiantajan määrittelemiä, projektin ulkopuolisia työtehtäviä projektin ohella neljänä arkipäivänä viikossa koko projektin ajan. Tämän lisäksi opinnäytetyön tekijä suoritti koulutusohjelman puuttuvat kurssit yhtäaikaisesti.

Ajanhallinnan aiheuttamien haasteiden lisäksi olemassa olevan sovellusalustatoteutuksen perustat aiheuttavat ongelmia sitä jatkokehitettäessä. Nykyinen toteutus on enimmäkseen prototyyppitasolla, joka puolestaan tarkoittaa sitä, ettei koodiperusta ole kovin helposti lähestyttävä saati jatkokehitettävä. Haasteista selviytyminen vaatii vankkaa tuntemusta erilaisista web-kehitykseen käytetyistä tekniikoista ja niiden hyödyntämisestä. Tekniikoiden lisäksi koodiperustan tuntemus helpottaa projektin haasteista selviytymistä huomattavasti.

1.2 Työn rakenne

Raportin luvussa 2 käsitellään projektin kannalta keskeisimmät tekniikat ja käsitteet. Projektin aikana esitellyt lyhenteet ja termit on avattu liitteessä 1.

Teoriaosuuden jälkeen esitellään produktin toimintaympäristönäkin toimivaa Deveota luvussa 3. Luku esittelee Deveon REST-ohjelmointirajapinnan, Deveo Web Clientin, Deveo Web Client -sovellukset ja niiden keskinäiset sidokset.

Raportin luvussa 4 esitellään projektille asetetut tekniset ja toiminnalliset vaatimukset. Luku käsittelee lyhyesti myös projektin vaatimusmäärittelyprosessia.

Vaatimusten jälkeen käydään projektissa syntyneitä tuloksia yksityiskohtaisesti läpi luvussa 5. Projektin tuloksien lisäksi luku kuvaa lyhyesti kehitysympäristön jossa produktia tuotettiin.

Lopuksi tehdään yhteenveto projektista kokonaisuutena luvussa 6. Luvussa kerrataan opinnäytetyön tulokset, tuodaan ilmi projektin aikana syntyneitä parannus- ja jatkokehitysehdotuksia sekä arvioidaan opinnäytetyön tekijän omaa oppimista.

2 Keskeiset tekniikat ja käsitteet

Tässä luvussa esitellään projektin kannalta keskeisimmät tekniikat ja käsitteet. Luku luo pohjan raportin myöhemmille luvuille esittelemällä produktin kannalta oleellisen REST-arkkitehtuurimallin. Lisäksi luku esittelee kehitysympäristössä käytetyn Vagrant-automaatiotyökalun sekä jatkokehitettävissä sovelluksissa käytetyt Ruby-pohjaiset web-sovelluskehykset Ruby on Railsin ja Sinatran opinnäytetyön kannalta mielekkäällä tavalla, niiden reititystoteutuksen osalta.

2.1 Vagrant

Vagrant on helppokäyttöinen automaatioon fokusoitunut työkalu kokonaisten kehitysympäristöjen luomiseen. Se madaltaa kehitysympäristöjen pystytysaikaa sekä lisää tuotanto- ja kehitysympäristöjen samankaltaisuutta. (Vagrant 2013.)

Vagrant mahdollistaa tehokkaan tavan virtualisoitujen kehitysympäristöjen luomiselle, hallinnoimiselle ja niiden kanssa työskentelemiselle ohjelmistoprojekteissa. Luomalla projektikohtaisia virtuaalisia ympäristöjä kyetään niiden riippuvuudet ja vaatimukset pitämään eristyksissä toisistaan. Eristyksen ansiosta projektikohtaisesti asennetut ohjelmistot eivät myöskään sotkeudu isäntäkoneen omien ohjelmistojen kanssa. Kehitysympäristöt saadaan tuntien konfiguroimisen ja ohjelmistojen asennuksen sijasta tehtyä yhdellä ainoalla komentorivikomennolla. Ympäristöjen poistaminen ja uudelleenpystyttäminen käy yhtä vaivattomasti. (Peacock 2013, 1.)

Vagrantin avulla (Peacock 2013, 7)

- voidaan tuotantoympäristöä imitoida kehitysympäristössä
- voidaan ympäristöjen päivittämiseen käytettyjen integroitujen varustustyökalujen (Chef ja Puppet) konfiguraatiot tallentaa standardoidussa formaatissa
- voidaan projektit jakaa omiin virtualisoituihin ympäristöihinsä, jolloin vältytään ympäristöjen konfiguraatioiden ja riippuvuuksien versioiden eroavaisuuksien aiheuttamilta ongelmilta
- uudet projektijäsenet saavat kehitysympäristön pystytettyä muutaman komennon avulla

- sanonta ”It works on my machine” tehdään tarpeettomaksi
- kehittäjät säästyvät isäntäkoneella tuotetun ohjelmakoodin ja virtuaaliympäristössä ajettavan sovelluksen linkittämisen aiheuttamalta päänvaivalta Vagrantin hoitaessa projektitiedostojen automaattisen synkronoinnin
- luodut kehitysympäristöt voidaan laittaa käyttäytymään paikallisen koneen tapaisesti kartoittamalla ympäristön web-palvelimen käyttämä TCP-portti 80 vastaavaan porttiin isäntäkoneella
- voidaan omaa kehitysympäristöä helposti sekä esitellä kollegoille että jakaa kollegoiden kesken.

Vagrant voidaan asentaa niin Mac OS X:ään, Linuxiin kuin Windowsiinkin ja koska se vaatii Rubyn toimiakseen, sisältää sen asennuspaketti sulautetun Ruby-tulkin. Ainoa toinen riippuvuus on jokin virtualisointityökalu, kuten esimerkiksi Oracle VirtualBox. (Peacock 2013, 8.)

2.2 REST

Representational State Transfer eli REST on ohjelmointirajapintojen toteuttamiseen käytetty arkkitehtuurimalli, jota yleisimmin käytetään yhdessä HTTP-protokollan kanssa. REST itsessään ei ole protokolla, tiedostoformaatti eikä ohjelmistokehys. Se tarjoaa joukon arkkitehtuurisia rajoitteita, jotka korostavat komponenttien vuorovaikutuksen skaalautuvuutta, rajapintojen yhdenmukaisuutta ja komponenttien keskinäistä riippumattomuutta. Näitä rajoitteita kutsutaan Fieldingin rajoitteiksi, sillä ensimmäisen kerran ne totesi Roy T. Fielding väitöskirjassaan vuonna 2000, kooten ne nimityksen ”REST” alle. REST-arkkitehtuurimallin periaatteita eli RESTful-periaatteita hyödyntävät ohjelmistot käyttävät HTTP-pyyntöjä datan lähettämiseen luonti- ja päivitysoperaatioissa, datan lukemiseen ja datan poistamiseen. (Elkstein 2008; Fielding 2000; Richardson & Amundsen 2013, 29.)

REST käsittää kaksi oleellista konseptia: resurssit ja representaatiot. Resurssi voi olla mikä tahansa itsessään tärkeä asia, johon halutaan pystyä viittaamaan. Yleisimmin resurssi on asia joka voidaan tallentaa tietokoneelle, kuten esimerkiksi dokumentti, rivi

tietokannassa tai jonkin ajetun algoritmin tulos. Resurssi voi kuitenkin olla mikä tahansa muukin asia, kuten esimerkiksi granaattiomena, ihminen, jokin väri, rohkeus käsitteenä, äidin ja tyttären välinen suhde tai kaikki alkuluvut sisältävä joukko. Resurssien ainoa rajoite on niiden tarve omata web-osoite, jotta ne kyetään identifioimaan ja jotta niihin kyetään viittaamaan. (Richardson & Amundsen 2013, 30.)

Yleisimmin resurssien tilaa kuvataan XML-dokumentin, JSON-olion, CSV-tiedoston tai sen luomisessa käytetyn SQL INSERT -lausekkeen muodossa. Näitä kuvauksia kutsutaan representaatioiksi. Resurssin representaatio voi olla mikä tahansa tietokoneen ymmärtämä dokumentti, joka voi sisältää mitä tahansa tietoa resurssista. Resurssilla voi olla monta eri representaatiota. Joillakin resursseilla voi olla esimerkiksi kaksi representaatiota, joista toinen on suurpiirteinen sisältäen vain vähän tietoa ja toinen tarkkapiirteinen sisältäen kaiken tiedon resurssin tilasta. Lisäksi jotkin ohjelmointirajapinnat tarjoavat saman tiedon eri formaateissa, esimerkiksi JSON- ja XML-muotoisena. (Richardson & Amundsen 2013, 30–32.)

Vaikka resurssit voivat olla mitä tahansa, ei niitä silti voi käyttää miten tahansa, sillä käsittelyyn on olemassa sääntöjä. RESTful-periaatteita noudattavissa järjestelmissä asiakkaat ja palvelimet ovat vuorovaikutuksessa toistensa kanssa ennalta määrätyn protokollan mukaisesti. Web-ohjelmointirajapintojen eli API:en maailmassa tämä kyseinen protokolla on HTTP. API-asiakkaat vuorovaikuttavat API:nsa kanssa lähettäen erityyppisiä HTTP-pyyntöjä. (Richardson & Amundsen 2013, 33.)

HTTP-standardi määrittelee kahdeksan erityyppistä HTTP-metodia, joista yleisimmin käytetyt ovat (Richardson & Amundsen 2013, 33):

- GET resurssin representaation pyytämiselle.
- DELETE resurssin poistamiselle.
- POST uuden resurssin luomiselle annetun representaation mukaisesti.
- PUT resurssin päivittämiselle annetun representaation mukaisesti.

Neljän yleisimmin käytetyn metodin lisäksi ohjelmointirajapinnan tutkimiseen käytetään kahta metodia (Richardson & Amundsen 2013, 33):

- HEAD jonkin resurssin representaation yhteydessä lähetettävien HTTP-otsikoiden pyytämiselle.
- OPTIONS jonkin resurssin hyväksymien HTTP-metodien selvittämiselle.

Näiden kuuden metodin lisäksi HTTP-standardi määrittelee metodit CONNECT ja TRACE, joita käytetään ainoastaan HTTP-välityspalvelimien kanssa, eivätkä ne täten ole oleellisia REST:n kannalta (Richardson & Amundsen 2013, 33).

2.3 Ruby on Rails

Ruby on Rails eli Rails on enemmän kuin pelkkä MVC-arkkitehtuurimallia hyödyntävä web-ohjelmistokehys. Rails on myös kehys web-sovellusten ajattelemiselle. Se tarjoaa sen, mitä useimmat käyttäjät useimmiten tarvitsevat tekemään useimpia asioita. Se auttaa keskittymään vain ja ainoastaan niihin asioihin joilla on merkitystä. (Fernandez 2010, xxxiii.)

Rails sisältää sisäänrakennetun reititysjärjestelmän, joka tarkastelee sovellukseen saapuvan HTTP-pyynnön web-osoitetta ja ottaa selville toimenpiteen, jonka sovelluksen olisi tarkoitus laittaa käytäntöön. Järjestelmän reitityssäännöt eli reitit määrittelevät hahmoja joiden avulla web-osoitteet voidaan tunnistaa ja joiden avulla web-osoitteita voidaan luoda. Hahmot voidaan luoda automaattisesti jonkin käytännön mukaisella tavalla, kuten esimerkiksi REST-resurssien tapauksessa. Hahmot voivat sisältää yhdistelmän staattisia alimerkkijonoja, web-osoitteen syntaksia imitoivia kauttaviivoja ja positioituja parametreja, jotka käyttäytyvät ikään kuin vastaavien web-osoitteen sisältämien arvojen vastaanottimina. Reitti voi myös sisältää kovakoodattuja positioituja parametreja avain-arvo -parien muodossa. Reititysjärjestelmä määrittelee itse kaksi kovakoodattua positioitua parametria jokaista reittiä kohden kontrollerin ja toimenpiteen tunnistamiselle. (Fernandez 2010, 31–32.)

Railsin reititysjärjestelmä sisältää paljon REST-arkkitehtuurimallia tukevia käytäntöjä. Mitä enemmän näitä käytäntöjä noudattaa, sitä kätevämmältä reititysjärjestelmä tuntuu. Rails voidaan laittaa luomaan automaattisesti standardit reitit jollekin tietylle resurssille. Tällöin Rails luo neljä nimettyä reittiä, yhteensä seitsemälle eri toimenpiteelle (Taulukko 1). (Fernandez 2010, 59–X.)

Taulukko 1. Esimerkki Railsin reititystoteutuksen standardeista REST-resurssikohtaisista reiteistä

HTTP-metodi	**Reitti**	**Toimenpide**
GET	/clients/1	show
GET	/clients	index
GET	/clients/1/edit	edit
GET	/clients/new	new
POST	/clients	create
PUT	/clients/1	update
DELETE	/clients/1	destroy

2.4 Sinatra

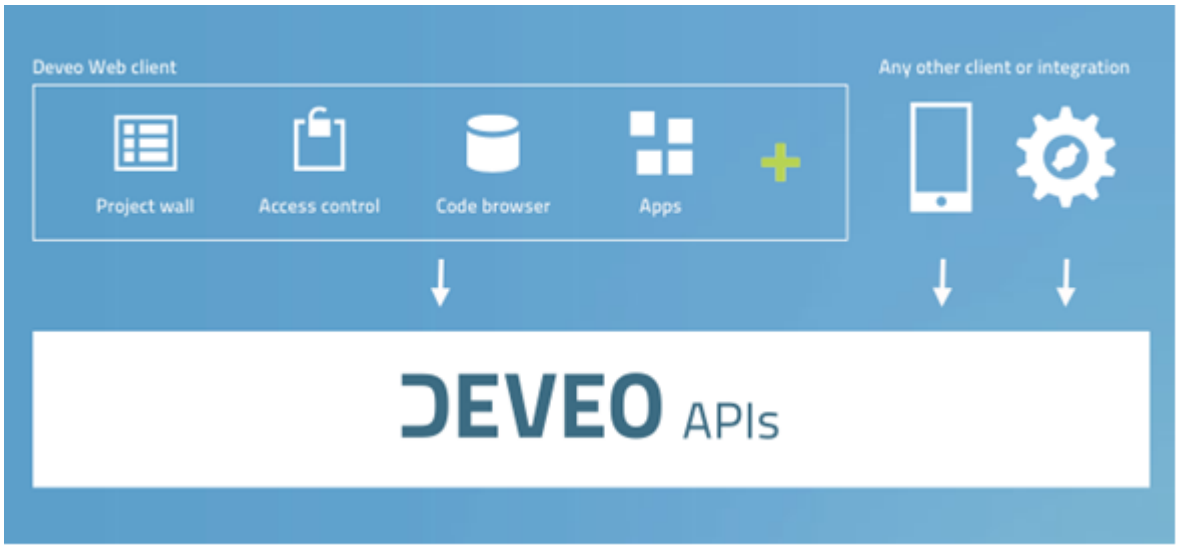
Sinatra ei varsinaisesti ole sovelluskehys, vaikka sitä yleensä sillä nimityksellä puhutellaankin. Sinatran mukana ei tule sisäänrakennettuja ORM-työkaluja, valmiita konfiguraatiotiedostoja, eikä edes projektihakemistoa, jollei sitä luo itse. Sinatra-sovellukset ovat luonnostaan erittäin joustavia eivätkä tyypillisesti ole yhtään sen suurikokoisempia kuin mitä tarvitaan. Sinatra ei pakota MVC-arkkitehtuurimallin tai minkään muunkaan toimintatavan käyttöä. Se on ainoastaan kevytrakenteinen Rack-väliohjelmiston ympärille toteutettu päällysohjelmisto, joka on ideaalinen valinta nopeaa kehitystä vaativille yksinkertaisille projekteille. (Harris & Haase 2011, 2–4.)

Railsin tapaan myös Sinatra tekee reititysmäärittelyistä helppoa. Reitit voivat yhtälailla sisältää positioituja parametreja, jotka sisällytetään toimenpiteen saamaan *params*-argumenttiin. Lisäksi reitin määrittelyssä voi käyttää jokerimerkkiä, joka asetetaan *params*-

argumentin *splat*-alkioon web-osoitteessa sijaitsevalla arvolla. Toisin kuin GET- ja DELETE-pyyntöjen yhteydessä, ei lähetystä vaativissa POST- ja PUT-pyynnöissä lähetetyn tietosisällön parametreja tarvitse määritellä erikseen web-osoitteessa, vaan ne sisällytetään niin ikään *params*-argumenttiin. Reittejä voidaan myös määritellä säännöllisten lausekkeiden avulla, jolloin useampi reitti osoittaa samaan toimenpiteeseen. (Harris & Haase 2011, 20–23.)

3 Deveo

Deveo koostuu kahdesta erillisestä sovelluksesta: Rails-pohjaisesta palvelinpäästä ja Sinatra-pohjaisesta selainpäästä. Selainpään toiminnallisuuden toteuttaa palvelinpäästä täysin irrallinen Deveo Web Client eli Client. Palvelinpää määrittelee Deveon tietomallit ja tarjoaa REST-ohjelmointirajapinnat eli API:t niiden käsittelemiseen. Palvelinpään ulkopuoliset sovellukset kykenevät käsittelemään tietomalleja API-toimintojen avulla. (Kuva 1.)



Kuva 1. Deveon arkkitehtuurikuvaus (Deveo 2014b)

3.1 REST-ohjelmointirajapinnat

Deveo mahdollistaa sen tietomallien käsittelyn julkisten API:en avulla. Jokainen tietomalli esitellään resurssina ja jokainen toteuttaa oman API:nsa. Yleisimmin nämä API:t käsittävät toiminnot uusien resurssien luomiselle, lukemiselle, päivittämiselle ja poistamiselle (Taulukko 2).

Taulukko 2. Deveon tietomallikohtaiset API-toiminnot (Deveo 2014a)

Tietomalli	**Luonti**	**Luku**	**Päivitys**	**Poisto**
Company		✓		
User	✓	✓	✓	✓
Group	✓	✓	✓	✓
Group Member	✓	✓	✓	✓
SSH key	✓	✓	✓	✓
Project	✓	✓	✓	✓
Project User	✓	✓	✓	✓
Project Group	✓	✓	✓	✓
Project Bot	✓	✓	✓	✓
Company Bot	✓	✓	✓	✓
Repository	✓	✓		✓
Hook	✓	✓	✓	✓
Hook Service		✓		
Commit	✓	✓		
Branch		✓		
Tag		✓		
Comment	✓	✓		
Event	✓	✓	✓	
Password Recovery	✓	✓	✓	
License		✓		

API:t hyödyntävät Railsin tehokasta reititystoteutusta ja vastaavat niihin lähetettyihin HTTP-pyyntöihin JSON-muotoisella tietosisällöllä, joka soveltuu erinomaisesti esimerkiksi JavaScript-pohjaisten web-sovellusten käytettäväksi. Jotta API-toimintoja kyetään käyttämään, tulee jokaisen toiminnon yhteydessä lähettää kolme API-avainta, jotta tehtyjen HTTP-pyyntöjen lähde kyetään tunnistamaan. API-asiakkaat saavat käsiinsä oman käyttäjätilin ja Deveo Companyn eli yrityksen API-avaimet luomalla uuden session. Sessio luodaan autentikoitumalla käyttäjän yrityksen uniikilla tunnisteella, käyttäjän sähköpostiosoitteella tai käyttäjänimellä ja salasanalla. Avaimet palautetaan onnistuneen autentikoinnin yhteydessä luodun session tiedoissa. Toimintojen yhteydessä välitettävä kolmas avain identifioi HTTP-pyynnön tekevän Deveo Pluginin eli liitännäisen. Jokainen toiminto kohdistetaan johonkin tiettyyn resurssiin jonkin tietyn yrityksen sisällä. (Deveo 2014a.)

Toiminnot auktorisoidaan käyttäjän HTTP Authorization -otsikossa lähettämien API-avainten avulla. Avainten avulla niin yritys, käyttäjä kuin liitännäinenkin pystytään tunnistamaan. Toiminnoille on ennalta määrätty sääntöjä, joiden perusteella vain tietyt käyttäjät saavat suorittaa tiettyjä toimintoja. Käyttäjille ennalta määrättyjen sääntöjen lisäksi liitännäisille tulee erikseen antaa lupa tiettyjen resurssien käsittelemiselle. Käytännössä tämä tarkoittaa sitä, että vaikka käyttäjällä olisikin oikeus suorittaa jokin tietty toiminto, voidaan suoritus silti evätä, mikäli liitännäiselle ei ole annettu tarvittavia oikeuksia, ja toisin päin. (Deveo 2014a.)

API:en resursseja esitellään ja käsitellään keskenään samankaltaisilla JSON-representaatioilla. Resurssien representaatiot käsittävät muutaman yhteisen attribuutin: API-statuksen (api_status), API-aikaleiman (api_timestamp), luontipäivämäärän (created_at) ja päivityspäivämäärän (updated_at). Lisäksi jokainen representaatio kuvaa resurssiaan monien muiden attribuuttien avulla. (Deveo 2014a.)

API:en lisäksi palvelinpää tarjoaa hallintapaneelin yhden Deveo-asennusinstanssin ylläpitoa varten. Hallintapaneelin kautta kyetään käsittelemään muun muassa asennusinstanssin lisenssiä, ylläpitäjiä, yrityksiä, sähköposti- ja autentikointiasetuksia sekä päivityksiä (liite 5). Palvelinpää ei asennusinstanssin hallintapaneelia lukuun ottamatta toteuta käyttöliittymää Deveoon, vaan jättää sen Clientin vastuulle.

3.2 Deveo Web Client ja Client-sovellukset

Deveon käyttöliittymän toteuttava Client on yksi palvelinpään liitännäisiä hyödyntävistä sen ulkopuolisista sovelluksista. Client tarjoaa Deveolle web-käyttöliittymän käyttäen palvelinpään API-toimintoja sisällön tuottamiseen, esittämiseen ja käsittelemiseen. Koska kaikki tieto on tallennettuna palvelinpäähän, Client on toteutukseltaan erittäin kevyt. Jokainen Deveon tietomalleihin kohdistunut toiminto suoritetaan Clientin aloitteesta palvelinpään API:en kautta palvelinpäässä. Client itsessään toteuttaa käyttöliittymän vain muutamille Deveon toiminnoille jättäen loput Client-sovellusten vastuulle. Keskeisimmät Clientin toteutukset ovat autentikoinnin käyttöliittymä, pääkäyttöliittymän valikot ja Client-sovellusten välillä navigointi. Client hyödyntää muutamaa avoimen lähdekoodin JavaScript-kirjastoa: jQuerya, Underscore.js:ää ja Modernizria. (Deveo 2014c.)

Client-sovellukset ovat kiinteä osa Clientin lähdekoodia vailla minkäänlaista sovelluksia yksilöivää toteutusta. Jokainen sovellus käyttää palvelinpään API-toimintoja Clientin oman liitännäisen oikeuksia hyödyntäen. Tämän takia Clientin liitännäiselle on myönnetty kaikki oikeudet, jotta jokainen sovellus sitä pystyisi hyödyntämään. Tästä johtuen jokaisella sovelluksella on oikeus tehdä mitä tahansa mille tahansa Deveo-resurssille, eikä API-toimintoihin tehtyjä HTTP-pyyntöjä pystytä yhdistämään tiettyihin sovelluksiin. Käytännössä tämä ei kuitenkaan ole ongelma, sillä lopulta jokainen sovellusten kautta suoritettu toiminto rajoittuu sovellusta käyttävän käyttäjän oikeuksiin.

Client-sovellusten vastuualueena osana Clientiä on luoda näkymiä Deveon eri tietomalleille. Kullakin Clientiä osana olevalla sovelluksella on vastuullaan jonkin tietyn Deveon tietomallin tai tietokokonaisuuden esittäminen, käsitteleminen tai molemmat. Mikään ei silti rajoita yhtä Client-sovellusta olla esittämättä tai käsittelemättä vaikka jokaista tietomallia. Rajana on ainoastaan se, mitä API-toimintoja palvelinpäällä on käyttäjilleen tarjota.

Client-sovellukset on toteutettu kokonaisuudessaan HTML:llä, CSS:llä ja JavaScriptillä. Client paketoi sovellusten JavaScript-koodit sovelluskohtaisiin olio-literaaleihin, jolloin

niissä kirjoitettu ohjelmakoodi suoritetaan sovelluksen omalla JavaScript-näkyvyysalueella eikä suoraan selaimen *window*-näkyvyysaluuella. Sovellusten JavaScript-näkyvyysalueella on käytettävissä Clientin puolesta ladatut avoimen lähdekoodin JavaScript-kirjastot. Avointen JavaScript-kirjastojen lisäksi käytettävissä on Deveon omia JavaScript-kirjastoja. JavaScript-kirjastojen ohella sovelluksilla on käytettävissään ennalta määrättyjä CSS-tyylejä Deveon oman CSS-tyylikehyksen muodossa. Tyylikehyksen valmiiden CSS-luokkien avulla sovellukset voidaan vaivattomammin tyylitellä näyttämään asianmukaisilta. (Deveo 2014c.)

Sovellukset koostuvat yhdestä tai useammasta näkymästä, joista kukin voidaan määritellä upotettavaksi Clientin käyttöliittymään kolmelle eri näkyvyysalueelle: yritys-, käyttäjä- tai projektinäkyvyysalueelle. Tieto siitä, mille näkyvyysalueelle mikäkin sovelluksen näkymä sijoittuu, määritellään sovelluksen omaan JSON-muotoiseen metatiedostoon avain-arvo -pareina. Tämän tekniikan johdosta voi käyttäjä Clientin yläpalkin navigaatiopainikkeiden avulla navigoidessaan nähdä yhden sovelluksen näkymiä jopa kolmella eri näkyvyysalueella. (Deveo 2014c.)

Näkyvyysalueen lisäksi näkymille voidaan valita yksi kahdesta esitystavasta: *Page View* eli sivunäkymä tai *Task View* eli tehtävänäkymä. Sivunäkymä upotetaan Clientin ulkoasuun siten, että sen toiminta-alueeksi muodostuu Clientin sivuvalikon ja sivun oikean reunan välinen alue (liite 2), kun taas tehtävänäkymä renderöidään sivunäkymän päälle ohuena kaistaleena sivun oikeasta laidasta (liite 3). Renderöintialueen lisäksi esitystavat erottuvat siten, että ainoastaan sivunäkymät otetaan huomioon Clientin sivuvalikkoa luotaessa. Sivunäkymät on mahdollista piilottaa sivuvalikosta määrittelemällä ne piilotetuiksi näkymän metatietoihin sovelluksen metatiedostoon. Metatietojen lisäksi sovellukset määrittelevät näkymissään käytetyt tekstit ja näkymien nimet merkkijonoina käännöstiedostoon, jokaista sovelluksen tukemaa kieltä kohden. (Deveo 2014c.)

4 Vaatimukset

Tässä luvussa perehdytään projektille ja produktille asetettuihin tavoitteisiin ja vaadittuihin toiminnallisiin ominaisuuksiin.

4.1 Tekniset vaatimukset

Projektin aikana produktin jatkokehitettäville komponenteille oli ennalta määritelty toteutukseen käytettävät tekniikat. Jatkokehitettävät komponentit tuli toteuttaa samoilla tekniikoilla kuin mitä niiden toteuttamiseen oli tähänkin asti käytetty: Ruby, JavaScript, HTML ja CSS.

Projektin aikana toteutettavista produktin komponenteista ainoa toteutustekniikan osalta määrittelemätön komponentti oli komentorivityökalu. Toteutustekniikan valitsemista rajoitti kuitenkin hieman se, että työkalun tuli toimia Apple Mac OS X, Linux ja Microsoft Windows käyttöjärjestelmissä.

Lisäksi produktin Clientiin toteutettavien käyttöliittymäkomponenttien tuli toimia moitteettomasti suosituimmilla moderneilla web-selaimilla kuten Google Chromella, Safarilla, Mozilla Firefoxilla, Operalla ja Microsoft Internet Explorerilla. Internet Explorerin osalta tuettaviksi versioiksi asetettiin versio 9 ja sitä uudemmat.

Alustavat ja suuntaa antavat projektille asetetut vaatimukset näyttivät seuraavilta:

- Toimeksiantajan ulkopuolisten toimijoiden tulisi kyetä luomaan uusia Client-sovelluksia komentoriviltä, kehittämään Client-sovelluksia paikallisesti ja määrittelemään kehitettyjen Client-sovellusten vaatimat oikeudet.
- Deveo-asennusinstanssin ylläpitäjien tulisi kyetä ottamaan Client-sovelluksia käyttöön ja poistamaan niitä käytöstä instanssikohtaisesti.
- Deveo-yritysten ylläpitäjien tulisi kyetä aktivoimaan ja deaktivoimaan Deveo-asennusinstanssissaan omassa yrityksessään käyttöönotettuja Client-sovelluksia.

- Palvelinpään tulisi toteuttaa ohjelmointirajapinta liitännäisten luomista, päivittämistä ja poistamista, sekä liitännäisten yrityskohtaista aktivoimista ja deaktivoimista, varten.
- Client-sovellusten tulisi käyttää omia liitännäisiään REST-ohjelmointirajapintakutsujen auktorisoimiseen.
- Clientin tulisi tukea Client-sovellusten käyttöönottoa.

4.2 Vaatimusmäärittely

Projektin vaatimusmäärittelyyn osallistui opinnäytetyön tekijän lisäksi toimeksiantajan muita työntekijöitä. Vaatimusmäärittelyprosessissa ensimmäinen askel oli tunnistaa tavoiteltu hyöty, jota kohti haluttiin pyrkiä. Hyödyn selvittyä aloitettiin sen tavoittamiseksi vaadittujen komponenttien ja toimintojen tunnistaminen.

Teknisten vaatimusten ohella projektille oli asetettu toiminnallisia vaatimuksia ominaisuuksien muodossa. Vaatimusmäärittelyssä pyrittiin siihen, ettei siinä määriteltäisi komponentteja, joiden tulisi toteuttaa halutut toiminnalliset vaatimukset, vaan valinnan vastuu jätettäisiin pääosin opinnäytetyön tekijälle. Lisäksi vaatimukset oli laadittu siten, että ne kuvastaisivat toiminnallisuuksia mahdollisimman korkealla tasolla.

5 Tulokset

Projektin tuloksena syntyi käyttäjän näkökulmasta yksinkertainen tapa luoda uusia Client-sovelluksia komentoriviltä ja ottaa ne käyttöön Clientin Apps-sovelluksen kautta. Tulos täytti produktille asetetut vaatimukset ja se otettiin tuotantokäyttöön produktin verifikaatioprosessin päätyttyä.

5.1 Kehitysympäristö

Kehitystyötä tehtiin Apple Mac OS X -käyttöjärjestelmän alaisuudessa, Vagrant-pohjaisessa virtuaaliohjelmointiympäristössä, jossa niin Deveon palvelinpäätä kuin Clientiäkin ajettiin Deveon pilvipalvelun tuotantopalvelinta muistuttavassa ympäristössä, Debian 6 -käyttöjärjestelmässä. Kehitystyössä Vagrant oli korvaamaton, sillä sen avulla kyettiin helposti luomaan tuotantopalvelimen kaltainen virtuaalikone, jossa produktia testattiin ja ajettiin. Tämän johdosta voitiin mitä todennäköisimmin olettaa, että tuleva toteutus tulisi toimimaan myös tuotantoympäristössä kehitysvaiheen päätyttyä.

Ohjelmakoodin tuottamiseen käytettiin Sublime Text 3 -tekstieditoria. Projektin lähdekoodia hallinnoitiin Git-versionhallintatyökalun avulla ja sitä hostattiin Deveossa. Web-kehityksessä HTML-rakennetta, JavaScript-koodia ja CSS-tyylejä debugattiin laajalti Google Chrome -selaimen DevTools-työkaluilla. Selainpään ohjelmakoodin toimivuutta testattiin projektin vaatimusten mukaisilla selaimilla. Ruby-ohjelmakoodin testaamiseen käytettiin RSpec-testityökalua, jonka avulla sovellusten testejä ajettiin Jenkinsillä Continuous Integration -käytännön mukaisesti. Projektin aikana jokaisen Deveossa hostatun projektin produktiin liittyneen repositoryyn tehdyn muutoksen yhteydessä ajettiin kussakin repositoryssa säilötyn ohjelmiston testit Jenkinsillä, jotta voitiin varmistua siitä, etteivät ohjelmakoodiin tehdyt muutokset vahingossa rikkoisi jo olemassa olevia toimintoja.

5.2 Deveo CLI

Uusien Client-sovellusten vaivatonta alustusta varten haluttiin kehittää komentorivityökalu, jolla tiedostojärjestelmään saataisiin vaivattomasti luotua yksinkertainen runko

Client-sovelluksesta, ja jolla kehitetty Client-sovellus saataisiin vaivattomasti paketoitua Clientin ymmärtämään muotoon.

Komentorivityökalun nimeksi annettiin Deveo CLI ja se toteutettiin Rubyn versiolla 2.0. Työkalu toteutettiin avoimen lähdekoodin Ruby-kirjaston, GLI:n, avulla ja se käsittää kolme komentorivikomentoa: *help*, *create* ja *package* (Taulukko 3). Työkalu on todettu toimivaksi niin Mac OS X:llä, Linuxilla kuin Windowsillakin.

Taulukko 3. Komentorivityökalun komennot

Komento	**Argumentit**	**Käyttötarkoitus**
help	-	Listaa käytettävissä olevat komennot
create	name	Luo uuden Client-sovelluksen annetulla nimellä
package	path	Paketoi olemassa olevan Client-sovelluksen annetusta polusta

Sovellusten rungon luomiseen käytetään komentorivityökalun sisään tallennettua valmista esimerkkisovellusta, joka kopioidaan aina *create*-komennon yhteydessä senhetkiseen hakemistoon argumenttina annetun nimen mukaiseen alihakemistoon. Kopioinnin ohessa esimerkkisovelluksen metatiedoston tietoihin päivitetään argumenttina annettu nimi. Nimen lisäksi metatietoihin generoidaan 32-merkkinen heksaluku, joka edustaa sovelluksen liitännäisen liitännäisavainta.

Sovellukset haluttiin paketoida mahdollisimman standardiin formaattiin, jotta paketointiprosessi voitaisiin suorittaa huoletta eri käyttöjärjestelmien alaisuudessa. Näillä perustein sovellusten pakettiformaatiksi valikoitui ZIP. Koska paketointi haluttiin tehdä alustariippumattomasti, suoritettiin se Ruby-ohjelmakoodilla avoimen lähdekoodin rubyzip-kirjaston avulla. Kirjasto pitää huolen siitä, että paketointi saadaan suoritettua kunkin käyttöjärjestelmän alaisuudessa ongelmitta.

Työkalu päätettiin julkaista MIT-lisenssillä Gemin muodossa Rubygems-hostauspalvelussa. Sovellus versioitiin semanttisen versioinnin mukaisella versiolla 0.0.1.

Jotta Client-sovelluksia kehittävien toimeksiantajan ulkopuolisten toimijoiden ei tarvitsisi ladata työkalua erikseen, haluttiin sitä levittää Clientin yhteydessä Gem-riippuvuutena. Riippuvuus määriteltiin Clientin Gemfileen siten, että se asennettaisiin ainoastaan kehitysympäristössä.

5.3 Deveo Plugins API

Deveon palvelinpäähän lisättiin API-toiminnot liitännäisten kokonaisvaltaista käsittelyä varten. Lisättyjen toimintojen yhteiseksi nimeksi annettiin liitännäisiä kuvaavan Plugin-tietomallin nimen mukaisesti Plugins API. Jotta uudet toiminnot voitiin lisätä, oli olemassa olevien toimintojen sijaintia muutettava. Olemassa olevista toiminnoista liitännäisten aktivointi ja deaktivointi Deveo-yrityksessä siirrettiin pois uusien päätepisteiden tieltä.

Ennen produktin toteutusta Deveo tarjosi API:n liitännäisten osalta ainoastaan yrityksen liitännäisten aktivoimiselle ja deaktivoimiselle. Kyseiset toiminnot suoritettiin suoraan Plugins API:n juureen, vaikka varsinaisesti toiminnot kohdistettiin Plugin-tietomallin sijasta Company-tietomalliin. Jotta väärinkäsityksiltä vältyttäisiin tulevaisuudessa, siirrettiin aikaisemmin toteutetut päätepisteet Deveo Companies API:n alapäätepisteiksi.

Plugin-tietomallille toteutettiin viisi eri päätepistettä liitännäisten eri toimintoja, luontia, lukua, muokkaamista ja poistamista, varten. Liitännäisiä luotaessa vaadituiksi parametreiksi asetettiin liitännäistunniste (id), nimi (name), liitännäisavain (plugin_key), versio (version) ja liitännäisen vaatimat oikeudet (permissions). Valinnaisiksi parametreiksi asetettiin liitännäisen kuvaus (description) ja tieto siitä, onko kyseessä Client-sovelluksen liitännäinen vai ei (external). Liitännäisiä päivittäessä mahdollisiksi päivitettäviksi kentiksi asetettiin nimi, versio, kuvaus ja liitännäisen vaatimat oikeudet. (Taulukko 4.)

Taulukko 4. Plugin-tietomallin päätepisteet suhteessa Deveon juureen. Taulukossa esitellyt päätepisteiden toiminnan kannalta pakolliset parametrit on merkitty tähdellä (*).

HTTP-metodi	**Päätepiste**	**Parametrit**
POST	/api/v0/plugins	id*, name*, plugin_key*, version*, description, external, permissions*
GET	/api/v0/plugins	-
GET	/api/v0/plugins/:id	-
PUT	/api/v0/plugins/:id	name, version, description, permissions
DELETE	/api/v0/plugins/:id	-

Liitännäisten luontiprosessissa otetaan huomioon kaikki Plugin-tietomallin validoinnit (Deveo 2013):

id

- Arvon tulee olla tietotyypiltään merkkijono.
- Arvo on pakollinen.
- Arvon tulee olla uniikki Deveo-asennusinstanssin näkyvyysalueella.
- Arvon minimipituus on 2 merkkiä.
- Arvon maksimipituus on 40 merkkiä.
- Arvo saa sisältää isoja ja pieniä kirjaimia väliltä a-z, numeroita väliltä 0-9 sekä viivoja (-) ja alaviivoja (_).

name

- Arvon tulee olla tietotyypiltään merkkijono.
- Arvo on pakollinen.
- Arvon minimipituus on 2 merkkiä.
- Arvon maksimipituus on 40 merkkiä.

description

- Arvon tulee olla tietotyypiltään merkkijono.

- Arvon maksimipituus on 512 merkkiä.
- Arvon oletusarvo on tyhjä merkkijono (””).

version

- Arvon tulee olla tietotyypiltään merkkijono.
- Arvo on pakollinen.
- Arvon tulee noudattaa semanttisen versioinnin versionumeron formaattia.

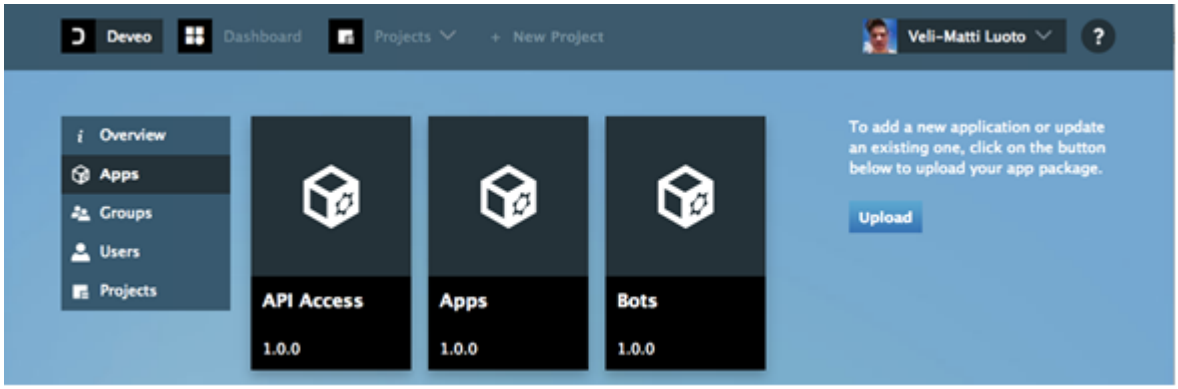
external

- Arvon tulee olla tietotyypiltään totuusarvomuuttuja.
- Arvon oletusarvo on epätosi.

Kaikkiin rajapinnan päätepisteisiin kohdennettuihin HTTP-pyyntöihin vastataan yhdenmukaisella Plugin-tietomallin JSON-representaatiolla (liite 4). Yksittäiset liitännäiset esitetään JSON-olion muodossa ja joukko liitännäisiä JSON-olioita sisältävän JSON-taulukon muodossa. Ainoa poikkeustapaus päätepisteiden palauttamassa representaatiossa on luontitoiminnon yhteydessä palautettavan representaation HTTP-statuskoodi 201, joka vastaa W3C:n määrittelemää HTTP/1.1:n Created-statusta (W3C 1999).

5.4 Deveo Web Client -sovellusten käyttöönotto

Produktin tuloksista näkyvin muutos on Client-sovellusten palvelimelle siirtoon tarkoitettu web-lomake, joka toteutettiin Clientin jo olemassa olleeseen Client-sovelluksia käsittelevään Apps-sovellukseen (Kuva 2). Sovellusten palvelimelle siirron yhteyteen niille toteutettiin sovelluspaketin validointi ja oman liitännäisen luonti. Palvelimelle siirron lisäksi Clientin sovellusarkkitehtuuria jatkokehitettiin siten, että sovellukset laitettiin auktorisoitumaan Clientin liitännäisen sijasta omien liitännäistensä tiedoilla API-toimintojen yhteydessä.



Kuva 2. Web-lomake Deveo Web Client -sovellusten käyttöönotolle

Sovellusten palvelimelle siirto haluttiin sallia ainoastaan Deveo-asennusinstanssin ylläpitäjille. Tämän mahdollistamiseksi toteutettiin siirto-operaation yhteyteen rajoitus siten, että Clientin palvelinpuoli pyytää kirjautuneen käyttäjän tiedot Deveo Users API:n kautta ennen operaation hyväksymistä. Jotta puutteellisilla oikeuksilla tehdyiltä siirtoyrityksiltä vältyttäisiin, on siirtoon tarkoitettu käyttöliittymän web-lomake näkyvissä ainoastaan asennusinstanssin ylläpitäjille.

Koska Clientin näkymäkoodista haluttiin siirtää dataa palvelinpuolelle, tarvittiin niiden välille uusi ohjelmointirajapinta. Rajapinnan tehtävänä oli mahdollistaa sovellusten siirto palvelimelle ja niiden poisto palvelimelta. Tätä varten oli Clientin palvelinpuolelle toteutettava kaksi uutta päätepistettä, toinen lisäykselle ja toinen poistolle. (Taulukko 5.)

Taulukko 5. Apps-päätepisteet suhteessa Clientin juureen. Taulukossa esitellyt päätepisteiden toiminnan kannalta pakolliset parametrit on merkitty tähdellä (*).

HTTP-metodi	**Päätepiste**	**Parametrit**
POST	/apps	application*
DELETE	/apps/:id	-

Siirto-operaatio käynnistetään Clientin Apps-sovelluksesta heti web-lomakkeen tiedostotyyppisen syötekentän arvon vaihtuessa eli silloin, kun käyttäjä on valinnut haluamansa tiedoston tiedostojärjestelmästään web-lomakkeen avulla. Siirtoon käytetään Clientin oman API-kirjaston upload-funktiota, joka pinnan alla suorittaa operaation

joko FormDatan tai iframe-tagin avulla selaimesta riippuen. Upload-funktio lähettää HTTP-pyynnön Clientin apps-päätepisteeseen POST-metodilla. Siirron onnistuessa kutsutaan näkymän render-olion uploadSuccessful-funktiota, joka renderöi näkymän Plugins API:n palauttaman datan mukaisesti. Mikäli siirto epäonnistuu, kutsutaan render-olion uploadUnsuccessful-funktiota, joka näyttää käyttäjälle tapahtunutta virhettä kuvastavan virheviestin. Näkymän virheviestit piilotetaan ennen jokaista operaatiota, jotta näkyvissä on aina ainoastaan uusimmat virheviestit. Lisäksi jokaisen operaation jälkeen operaation käynnistämä syötekenttä korvataan itsensä kopiolla käyttämällä jQueryn clone-funktiota siten, että sille annetaan ensimmäisenä argumenttina tosiarvo, jolloin kopiointiin ei oteta mukaan alkuperäisen elementin dataa eikä sidoksia.

Jotta käyttäjiltä voitaisiin evätä mielivaltaisten ZIP-pakettien siirtäminen palvelimelle sovelluslomaketta käyttäen, oli käyttäjän syöte validoitava. Tähän tarkoitukseen toteutettiin validaattori, jonka vastuulla on tarkistaa sovelluksen koko, metatiedot ja ohjelmakooditiedostot.

Sovellusten koon tarkistus toteutettiin laajentamalla Rubyn natiivia Dir-toteutusta siten, että sille lisättiin luokkametodi *size*, joka laskee annetusta hakemistopolusta rekursiivisesti jokaisen tiedoston koon lopulta palauttaen niiden summan annetun hakemistopolun kokona. Sovellusten maksimikooksi asetettiin 50 megatavua, jonka todettiin olevan enemmän kuin tarpeeksi ottaen huomioon sen, että Client-sovellukset koostuvat suurimmaksi osaksi erimuotoisista tekstitiedostoista.

Jotta koon vertailusta saatiin ohjelmakoodissa helppolukuista, päätettiin Dir-toteutuksen lisäksi laajentaa toista Rubyn natiivia toteutusta, Fixnumia. Fixnumiin lisättiin instanssimetodi *to_mb*, joka muuntaa tavut megatavuiksi, lisäten puolestaan ohjelmakoodin luettavuutta ja ymmärrettävyyttä. Laajennuksen myötä numeroita kyettiin vertailemaan megatavuina ja mahdollisessa virhetilanteessa sovelluksen koko voitiin palauttaa käyttäjälle vaivattomasti tavujen sijasta niin ikään megatavuina.

Tärkeimmäksi ja monimutkaisimmaksi toteutetuista validoinneista muodostui metatietojen validointi. Tärkeimmän siitä teki se, että mahdolliset virheet sovelluksen metatie-

doissa saattaisivat potentiaalisesti aiheuttaa virhetilanteen itse Clientin sovelluksia käsittelevässä ohjelmakoodissa. Monimutkaisuutta validointiin lisäsi metatietojen moninaisuus. Koska sovellusten metatiedosto toimi koko sovelluksen perustana ja koska siinä määriteltyjä sovellustietoja käytettiin sovellusten liitännäisten luontiin, oli tietojen oikeellisuus tärkeää tarkistaa. Metatietojen validoinnin ensimmäisessä vaiheessa tarkastettiin itse metatiedoston olemassaolo, joka puuttuessaan julistaisi koko sovelluksen virheelliseksi jo validoinnin alkuvaiheessa. Tämän jälkeen tarkastettiin metatietojen pakolliset jäsenmuuttujamäärittelyt, niiden tyypin oikeellisuus ja mahdolliset virheet niiden arvojen formaatissa. Kun pakolliset jäsenmuuttujatmäärittelyt oli todettu valideiksi, oli seuraava askel tarkastella niiden arvoja syvemmässä merkityksessään.

Clientin päässä validoituihin arvoihin kuuluivat näkymänimimäärittelyt sekä näkymien ulkoasu- ja näkyvyysaluemäärittelyt. Loput validoinneista jätettiin palvelinpään nojaan, joka validoi luotavaan liitännäiseen käytetyt meta-arvot. Koska metatiedostoon määriteltyjen näkymänimien oli tarkoitus vastata sovellukseen toteutettujen näkymien hakemistonimiä, tuli hakemistojen olemassaolo tarkastaa. Myös näkymänimien formaatti validoitiin, sillä niiden oli syytä sisältää ainoastaan tiedostojärjestelmätasolla hakemistoille sallittuja merkkejä. Metatiedoista viimeisimpänä validoitiin näkymien ulkoasutyyppi ja näkyvyysalue.

Kaikista validoinneista viimeisimpänä suoritettiin ohjelmakooditiedostojen sisällön oikeellisuuden tarkastaminen kullekin tiedostotyypille sopivimmalla Ruby-pohjaisella avoimen lähdekoodin kääntäjällä, renderöijällä tai parserilla. JavaScript-tiedostot käännettiin Uglifier-kompressorilla, CSS-tiedostot renderöitiin Sass-moottorilla ja JSON-tiedostot parsittiin Rubyn standardin kirjaston JSON-moduulilla.

Jotta sovellukset kykenisivät auktorisoitumaan palvelinpään API-toimintojen yhteydessä omana itsenään, oli niille palvelimelle siirron yhteydessä luotava oma liitännäinen. Koska kaikki sovelluksiin liittyvä käsittely tehtiin Clientin palvelinpuolen ohjelmakoodissa ja käyttöliittymäkoodi haluttiin pitää mahdollisimman yksinkertaisena, päätettiin myös rajapintakutsut suorittaa palvelinpuolelta, Ajax-kutsujen sijasta. Tämä oli ensimmäinen kerta kun Clientin palvelinpuolelta haluttiin käyttää palvelinpään API-toimintoja, sillä tähän astisista API-kutsuista kaikki oli tehty suoraan JavaScript-ohjelmakoodista Ajax-kutsuina. Käytännössä tämä tarkoitti sitä, että Clientin palvelinpuolelle oli

syytä toteuttaa yksinkertainen API-asiakasohjelma toimintojen käyttämistä varten. API-asiakasohjelman toteutuksessa hyödynnettiin vapaan lähdekoodin Ruby-pohjaista HTTP-asiakasohjelmaa Faradayta.

API-asiakasohjelmaan toteutettiin kuusi toimintoa:

- Liitännäisten luonti.
- Tietyn liitännäisen tietojen lukeminen.
- Liitännäisten muokkaaminen.
- Liitännäisten poisto.
- Yrityksen liitännäisten tietojen lukeminen.
- Käyttäjän tietojen lukeminen.

Näiden toimintojen avulla kyettiin liitännäisiä käsittelemään kokonaisvaltaisesti Clientin palvelinpuolelta käsin. Liitännäisten käsittelyyn liittyvistä toiminnoista API-asiakasohjelman osalta pois jätettiin liitännäisten yrityskohtainen aktivointi ja deaktivointi, sillä ne olivat jo toteutettuina Clientin Apps-sovelluksessa Ajax-kutsuin.

Käyttäjän tiedot tarvittiin käyttäjän auktorisoimiseen sovellusten palvelimelle siirron yhteydessä, sillä ainoastaan asennusinstanssin ylläpitäjille haluttiin antaa oikeus suorittaa kyseinen toimenpide. Jotta tietyn käyttäjän tiedot oli mahdollista pyytää API-asiakasohjelman avulla Deveo Users API:sta, piti käyttäjän käyttäjänimi olla tiedossa. Käyttäjänimi saatiin Clientin JavaScript-koodissa kirjautumisen yhteydessä asettamasta evästeestä, johon kirjautuneen käyttäjän käyttäjänimi oli tallennettuna. Evästeestä luku saatiin tehtyä vaivattomasti Ruby-ohjelmakoodista käsin Sinatran Cookies-laajennuksella.

6 Yhteenveto

Tässä luvussa käsitellään opinnäytetyön tuloksia ja projektin aikana opinnäytetyön tekijälle ja toimeksiantajalle ilmenneitä jatkokehitys- ja parannusehdotuksia. Lopuksi käydään läpi opinnäytetyön tekijän projektin aikana hankittua oppimista ja sen arviointia.

6.1 Opinnäytetyöprojektin tulokset

Deveo Oy:n toimeksiannosta opinnäytetyönä läpiviedyn projektin tarkoituksena oli jatkokehittää Deveo-sovellusalustan sovellustukea. Tuotetun produktin tuli toteuttaa vaatimusmäärittelyssä esitellyt käyttötarpeet pienimmällä mahdollisella panoksella siten, että produktista olisi kuitenkin mahdollisimman paljon hyötyä ja se voitaisiin ottaa tuotantokäyttöön sen valmistuttua. Edellä mainitut vaatimukset täytettiin ja toteutus siirtyi tuotantokäyttöön toimeksiantajan määrittelemän verifikaatioprosessin kautta.

Projektin aikana toteutettiin seuraavat asiat:

- Deveon palvelinpäähän lisättiin API-toiminnot Plugin-tietomallin kokonaisvaltaista käsittelyä varten.
- Client-sovellusten rungon luomista ja valmiiden Client-sovellusten paketointia varten toteutettiin alustariippumaton komentorivityökalu.
- Clientin palvelinpuolelle lisättiin rajapinta Client-sovellusten hallitsemista varten.
- Clientin palvelinpuolelle toteutettiin API-asiakasohjelma palvelinpään API-toimintojen käyttämistä varten.
- Clientin Apps-sovellukseen lisättiin web-lomake Client-sovellusten käyttöönottoa varten.
- Clientin sovellusarkkitehtuuria jatkokehitettiin siten, että jokainen Client-sovellus auktorisoituu Clientin sijasta omana itsenään palvelinpuolen API-toimintojen yhteydessä.

Projektin tuloksista jatkoa ajatellen tärkeimmät toteutukset ovat palvelinpäähän lisätyt API-toiminnot liitännäisten hallinnointia varten sekä Clientin sovellusarkkitehtuurin auktorisointitoteutuksen jatkokehitys. Lisätyt toiminnot mahdollistavat liitännäisten

hallinnoinnin palvelinpään ulkopuolella, joka mahdollistaa muun muassa liitännäisiä käsittelevien Client-sovellusten kehittämisen. Sovellusarkkitehtuurin auktorisointitoteutuksen jatkokehitys mahdollistaa muun muassa API-toimintoihin kohdistettujen sovelluskohtaisten HTTP-pyyntöjen tunnistamisen. Opinnäytetyön kannalta merkittävin tulos oli tuki Client-sovellusten käyttöönotolle.

Tuloksista hyötyjen kannalta turhimmaksi toteutetuksi komponentiksi todettiin komentorivityökalu Deveo CLI. Työkalulla ei projektin tulosten tuottamia hyötyjä ajatellen ollut juuri minkäänlaista merkitystä. Työkalun hyödyt todettiin niin minimaalisiksi, ettei sitä koskaan otettu osaksi Client-sovellusten kehitystyötä. Toteutus loi silti helposti jatkokehitettävän pohjan monikäyttöiselle komentorivityökalulle.

6.2 Jatkokehitys- ja parannusehdotukset

Projektin edetessä ilmeni lukuisia jatkokehitys- ja parannusehdotuksia, joista jokainen ehdotus jätettiin projektin aikana toteuttamatta. Deveon toimintatapoja mukaillen hyödyllisimmät ja ilmeisimmät ehdotukset on esitelty tässä alaluvussa.

Deveon tietomalleihin liittyvistä toiminnoista on tapana jäädä jälki järjestelmään tapahtumien muodossa. Jotta sovellusten elinkaaresta jäisi järjestelmään pysyvämpi jälki, voitaisiin niin ikään sovelluksen elämän eri vaiheista luoda tapahtumia. Tässä mielessä mielekkäitä tapahtumia olisivat mm. sovellusten asennus-, päivitys- ja poisto-toiminnot. Tämän tyylinen datan keruu mahdollistaisi sovelluksiin liittyvien tapahtumien esittämisen esimerkiksi Clientin yritysnäkyvyysalueelle sijoitetun sovelluksen tapahtumalokin muodossa. Sovellusten tapahtumaloki voisi antaa paremman käsityksen mm. sovellusten stabiiliudesta ja päivitystiheydestä.

Client-sovellukset jaetaan kahteen eri kategoriaan: mukana toimitettaviin core-sovelluksiin ja jälkeenpäin asennettaviin noncore-sovelluksiin. Vaikka core-sovellukset eivät toteutukseltaan eroa millään tavalla noncore-sovelluksista, käsitellään niitä silti pinnan alla hieman eri tavoin. Sovellukset ovat tallennettuina tiedostojärjestelmään eri hakemistoihin, koska core-sovellukset haluttiin eriyttää noncore-sovelluksista niiden toimitustavan

eroavuuden takia. Tarve eri tiedostojärjestelmäsijainnille voitaisiin eliminoida esimerkiksi siten, että core-sovellukset tallennettaisiin Clientin sijasta omiin repositoryihinsa ja ladattaisiin niistä Deveon asennuksen yhteydessä samaan hakemistoon kuin mihin noncore-sovelluksetkin ladataan. Toisin kuin noncore-sovellusten osalta, ei core-sovelluksia ole mahdollista poistaa tai edes deaktivoida. Pääsyy core-sovellusten poiston ja deaktivoimisen eväämiseen on se, että niiden ja Clientin välillä on riippuvuuksia. Riippuvuuksista eroon pääseminen on suurempi operaatio, joka puolestaan vaatii huomattavasti enemmän työtä.

Sovellusten päivityksen yhteydessä on melko yleistä, että uusien ominaisuuksien ja bugikorjauksien ohella päivitykset pitävät sisällään myös uusia bugeja. Tätä ongelmaa voitaisiin lievittää siten, että kaikki asennusinstanssiin asennetut sovellusversiot säilytettäisiin ja niitä käyttäen toteutettaisiin sovelluksille downgrade-toiminto, jolla voitaisiin palata käyttämään aikaisemmin asennettuja, toimivaksi todettuja versioita.

Sovellukset on alusta alkaen versioitu semanttisen versioinnin mukaisesti, mutta versiointia ei ole varsinaisesti hyödynnetty millään tavalla missään vaiheessa. Yksi mahdollinen hyödyntämiskohde olisi sovellusten versiokohtaisen changelogin ylläpitäminen. Sovellusten kehittäjät voisivat sen avulla kätevästi kirjata ylös muuttuneet asiat päivitysten yhteydessä esimerkiksi sovelluksen mukana toimitettavan tekstitiedoston muodossa, jota Client voisi myöhemmin hyödyntää Apps-sovelluksessaan sovelluksen tietojen esittämisen yhteydessä.

6.3 Oman oppimisen arviointi

Aihealueena opinnäytetyö oli laaja ja sen yhdeksi hankalimmaksi osa-alueeksi muodostuikin aihealueen rajaaminen. Aihealue saatiin lopulta kuitenkin rajattua projektin kannalta mielekkääksi. Eniten hankaluuksia opinnäytetyössä aiheutti opinnäytetyöraportin kirjoittaminen, sillä aikaisempaa kokemusta vastaavien raporttien työstämisestä, tai asiatekstien kirjoittamisesta ylipäänsä, oli opinnäytetyön tekijällä erittäin vähän. Tämän lisäksi ei raportin rakenteelle ollut olemassa selvää vakiintunutta mallia, joka hankaloitti raportin rakenteen muodostamista. Raportintyöstämisongelmien ehkäisemiseksi olisi

kokemusta raporttien kirjoittamisesta voitu hankkia keskittymällä syvemmin opintojen aikana aikaisemmin kirjoitettuihin raportteihin.

Projekti oli luonnollisesti hyvin työelämälähtöinen, sillä se suoritettiin työelämän yhteydessä. Tämä ei kuitenkaan välttämättä ollut ainoastaan hyvä asia, sillä opintojen ja työelämän sekoittaminen keskenään tuo mukanaan omat ongelmansa. Työtehtävien ja toimeksiantajan määrittelemien projektille asetettujen tehtävien keskenään priorisoiminen oli aika-ajoin hankalaa. Jälkikäteen ajateltuna projekti olisi voinut olla mielekkäämpi toteuttaa ja aikataulullisesti helpompi käsitellä omien työtehtävien ulkopuolella täysin eri kontekstissa, jolloin priorisointia olisi voitu tehdä oman arviointikyvyn mukaisesti. Hyviä puolia työelämän yhteydessä suoritetussa produktityyppisessä projektissa on se, ettei työtä joudu tekemään ilmaiseksi. Lisäksi produktia toteutettaessa on toimeksiantajan puolelta läsnä hyvänlaatuinen paine produktin valmiiksi saamiseksi, joka puolestaan lisää projektin tekijän motivaatiota.

Opinnäytetyöhön käytetty työmäärä ei poikennut merkittävästi siihen suunnitellusta työmäärästä, mutta aikataulullisesti projekti venähti useammalla kuukaudella. Pääsyynä projektin aikataulun venymiselle oli ajan puute, joka olisi pitänyt ottaa paremmin huomioon projektin aikataulua suunnitellessa. Koska suurin osa siitä ajasta, joka oli suunniteltu projektin työstämistä varten, käytettiin joko projektin ulkopuolisten työtehtävien tai opintosuoritusten suorittamiseen, oli lopputulos odotettavissa. Aikataulun venymisen aiheuttaja ei suinkaan ollut itse produktin toteutusvaihe vaan opinnäytetyöraportin työstämisvaihe. Projektin aikana toteutettu produkti saatiin valmiiksi aikataulusta edellä vuoden 2013 joulukuun lopusta poiketen jo marraskuun lopulla.

Lähteet

Deveo 2013. Deveo Developer Documenation. Luettavissa: https://developer.deveo.com/blog/docs/api/plugin/. Luettu: 2.4.2014.

Deveo 2014a. Deveo API Documentation. Luettavissa: https://developer.deveo.com/docs/api/. Luettu: 30.4.2014.

Deveo 2014b. Deveo – Features – Extensibility. Luettavissa: https://deveo.com/features/extensibility. Luettu: 30.4.2014.

Deveo 2014c. Web Client. Platform. Wiki. Web Client. Luettavissa: https://deveo.com/client. Luettu: 24.4.2014.

Deveo 2014d. What is Deveo?. Luettavissa: https://deveo.com/. Luettu: 17.2.2014.

Elkstein, M. 2008. What is REST?. Luettavissa: http://rest.elkstein.org/2008/02/what-is-rest.html. Luettu: 23.4.2014.

Fernandez, O. 2010. The Rails 3 Way. Addison-Wesley. Boston.

Fielding, R. 2000. Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). Luettavissa: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Luettu: 23.4.2014.

Harris, A. & Haase, K. 2011. Sinatra: Up and Running. O'Reilly. Sebastopol.

Mozilla Foundation. FormData. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/FormData. Luettu: 9.4.2014.

Neukirchen, K. 2012. Rack: a Ruby Webserver Interface. Luettavissa: http://rack.github.io. Luettu: 3.5.2014.

Peacock, M. 2013. Creating Development Environments with Vagrant. Packt Publishing. Birmingham.

Richardson L. & Amundsen, M. 2013. RESTful Web APIs. O'Reilly. Sebastopol.

ThoughtWorks 2014. Continuous Integration. Luettavissa: http://www.thoughtworks.com/continuous-integration. Luettu: 9.4.2014.

Vagrant 2013. About Vagrant. Luettavissa: http://www.vagrantup.com/about.html. Luettu: 18.2.2014.

W3C. What is CSS?. Luettavissa: http://www.w3.org/Style/CSS/Overview.en.html. Luettu: 9.4.2014.

W3C 1999. HTTP/1.1: Status Code Definitions. Luettavissa: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. Luettu: 9.4.2014.

W3Schools 2014a. HTML iframe tag. Luettavissa: http://www.w3schools.com/tags/tag_iframe.asp. Luettu: 9.4.2014.

W3Schools 2014b. XML Introduction – What is XML?. Luettavissa: http://www.w3schools.com/xml/xml_whatis.asp. Luettu: 3.5.2014.

Liitteet

Liite 1. Lyhenteet ja termit

API (Application Programming Interface)

API on lyhenne englanninkielisestä vastineesta ohjelmointirajapinnalle. Sitä käytetään yleisesti lyhempänä tapana viitata ohjelmointirajapintoihin.

Changelog

Changelog on nimitys, jota käytetään sovellusten versioiden välillä tapahtuneiden muutosten ylös kirjaamiseen.

CI (Continuous Integration)

CI on ohjelmistokehitystyössä käytetty käytäntö, joka perustuu jaetun ohjelmakoodin muutoksien yhteydessä ajettaviin automatisoituihin vahvistusajoihin, jonka avulla ohjelmakoodin ongelmat löydetään nopeasti (ThoughtWorks 2014).

CSS (Cascading Style Sheets)

CSS on yksinkertainen mekanismi, jonka avulla voidaan lisätä tyylejä kuten värejä ja fontteja, web-sivuille (W3C).

Downgrade

Downgrade on käsite, joka tarkoittaa jonkin ohjelmiston tai laitteiston palauttamista vanhempaan versioon. Operaationa downgrade on päivittämisen vastakohta.

FormData

FormData on JavaScript-rajapinta, joka tarjoaa helpon tavan HTML-lomakkeen kenttien esittämiselle avain-arvo -pareja sisältävän tietojoukon muodossa (Mozilla Foundation).

Gem

Gem on RubyGems-paketinhallintajärjestelmän formaatti Ruby-ohjelmien ja -kirjastojen jakeluun.

Gemfile

Gemfile on RubyGems-paketinhallintajärjestelmän projektikohtaisten Gem-riippuvuuksien määrittelemiseen käytetty tiedosto.

Git

Git on laajasti käytetty hajautettu versionhallintajärjestelmä.

HTML (HyperText Markup Language)

HTML on kuvauskieli, jolla yleensä muodostetaan web-sivujen rakenne.

HTTP (HyperText Transfer Protocol)

HTTP on selainten ja WWW-palvelimien väliseen tiedonsiirtoon käytetty protokolla.

iframe

iframe on HTML-tagi, jota käytetään toisen HTML-dokumentin upottamiseen isäntädokumentin sisälle (W3Schools 2014a).

JavaScript

JavaScript on skriptauskieli, joka suoritetaan selaimessa ja jota yleensä käytetään dynaamisen toiminnallisuuden luomiseen web-sivuilla.

jQuery

jQuery on JavaScript-kirjasto, joka on suunniteltu yksinkertaistamaan JavaScriptin moninaisia funktioita.

JSON (JavaScript Object Notation)

JSON on merkintäkieli, jolla voidaan kuvata tietorakenteita.

Modernizr

Modernizr on pieni JavaScript-kirjasto, jota käytetään erilaisten natiivien uuden sukupolven web-toteutusten tunnistamiseen.

MVC (Model-View-Controller)

MVC on arkkitehtuurimalli, joka jakaa sitä toteuttavan sovelluksen kolmeen keskenään yhdistettyyn osaan: malliin (Model), näkymään (View) ja ohjaimeen (Controller).

ORM (Object-Relational Mapping)

ORM on ohjelmointitekniikka, joka muuntaa relaatiotietokantojen datan oliopohjaisten ohjelmointikielten käytettäväksi.

Rack

Rack tarjoaa minimaalisen rajapinnan Rubya tukevien web-palvelimien ja Ruby-pohjaisten web-sovelluskehysten välille (Neukirchen 2012).

RSpec

RSpec on Ruby-pohjainen testikehys.

Ruby

Ruby on dynaaminen olioperustainen ohjelmointikieli.

Spin-off

Spin-off on eräänlainen yhtiötoimenpide, joka kuvastaa yhtiön jaoston irtautumista uudeksi erilliseksi yhtiöksi.

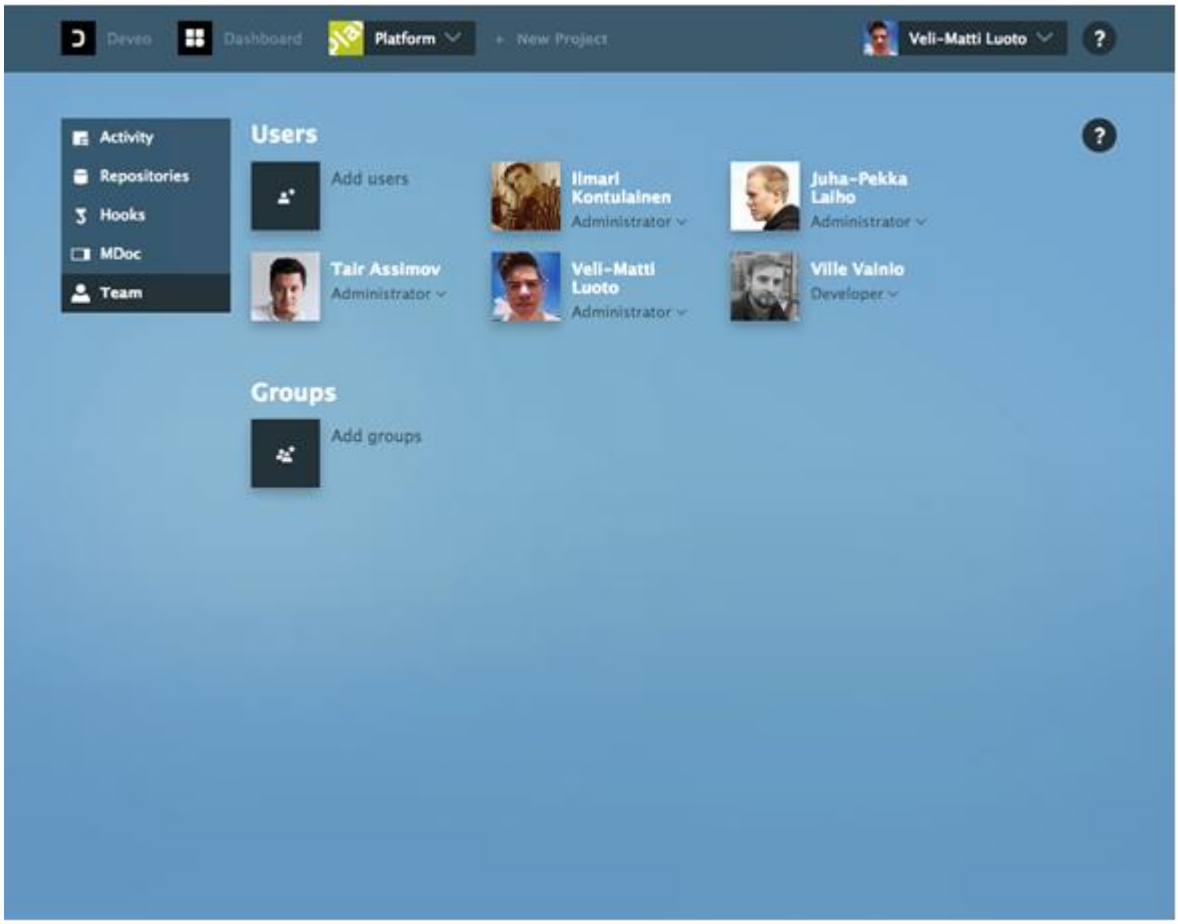
Underscore.js

Underscore.js on JavaScript-kirjasto, joka tarjoaa käyttäjälleen funktiot, jotka olettaisi kielessä olevan ja omasta takaa.

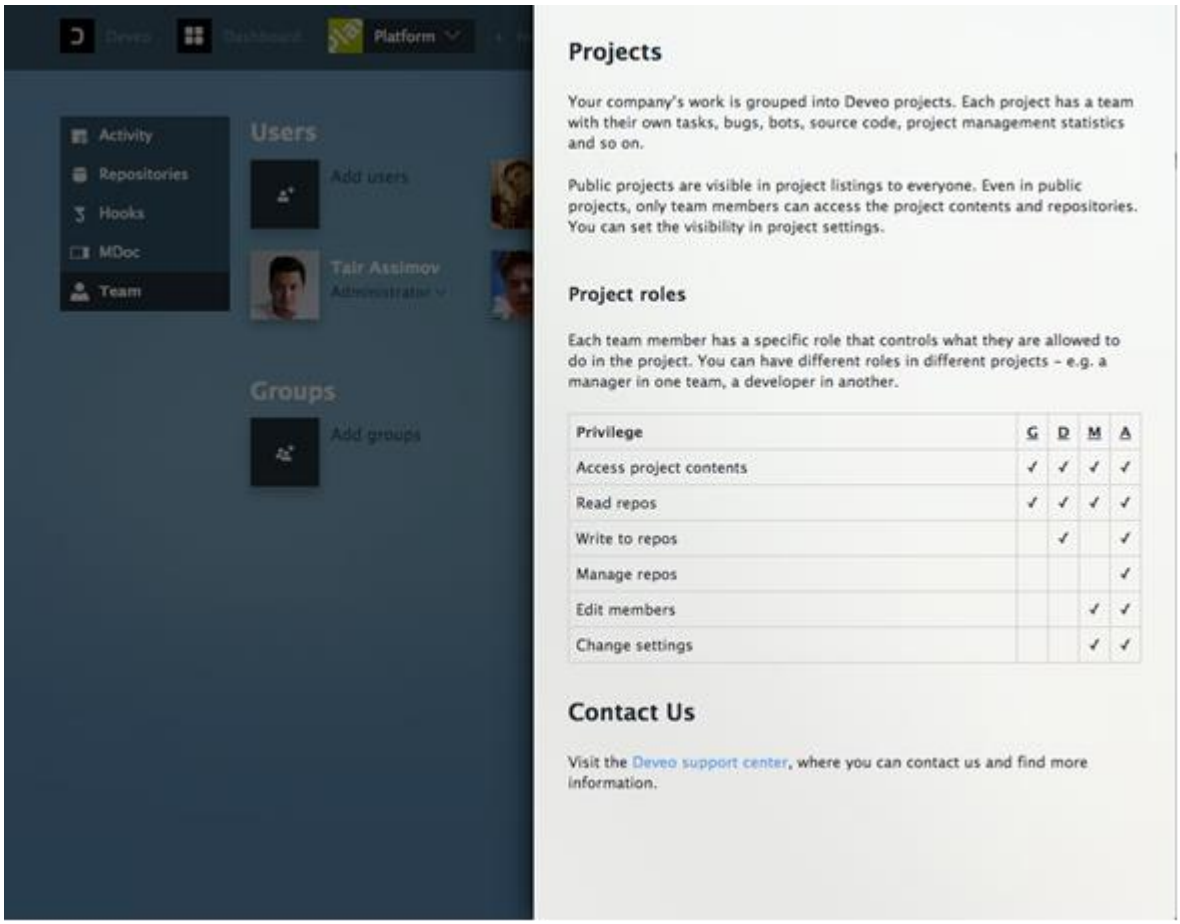
XML (Extensible Markup Language)

XML on tiedon tallentamiseen ja siirtämiseen suunniteltu kuvauskieli (W3Schools 2014b).

Liite 2. Deveo Web Client -sovelluksen sivunäkymä



Liite 3. Deveo Web Client -sovelluksen tehtävänäkymä



Liite 4. Plugin-tietomallin JSON-representaatio


```
{
  "api_status": 200,
  "api_timestamp": "2014-04-09T11:07:31Z",
  "id": "code",
  "created_at": "2013-12-30T13:41:34Z",
  "updated_at": "2014-03-25T16:21:31Z",
  "name": "Code",
  "description": null,
  "version": "1.0.0",
  "core": true,
  "external": false,
  "keys": {
    "plugin_key": "c7c0c114ed2a8f9444269762da0eedc2",
    "company_key": "cefa37e71e027dddb65407000080bebd",
    "account_key": ""
  },
  "activated": true,
  "permissions": {
    "create": [
      "repositories"
    ],
    "read": [
      "repositories",
      "commits",
      "project_users"
    ],
    "update": [
      "repositories"
    ],
    "delete": [
      "repositories"
    ]
  }
}
```

Liite 5. Kuvankaappaus Deveo-asennusinstanssin hallintapaneelista